

Cloud Services vs. Local Clusters

Andrew Lee, William Krzyzkowski, Brendan Armani

April 2019

1 Background

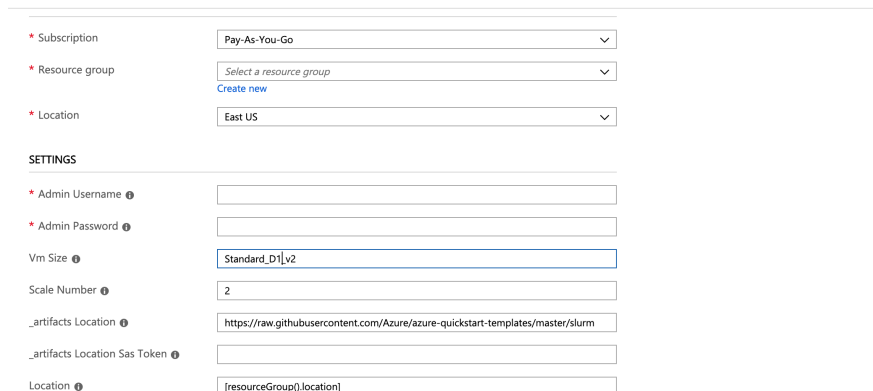
A distributed system consists of several machines networked together in some fashion. A High Performance Cluster (HPC) is a special distributed system designed to optimize running jobs that may involve communication among all nodes. It often uses a job scheduler such as SLURM to do this. JMU owns a HPC that is currently located in ENGEO. We investigated what it would take to make a HPC in three cloud platforms, and then performed cost and performance analysis on all of them. A distributed system in the cloud is one in a datacenter potentially far away from its users that is communicated with through a website provided by the cloud service provider. These clusters are configured in ways that are opaque to the user, and may have performance impacts.

Our research aims to evaluate which system, the cloud or a local HPC cluster, should be used for a class such as CS470 in today's day and age. We focus on performance, cost, and ease of use to come to a conclusion on this. We also compare the different cloud services we used against each other to see which would be the best suited for this situation. We investigated Google Cloud Platform (GCP), Amazon Web Services (AWS) and Microsoft Azure.

2 Methods

2.1 Microsoft Azure

In order to set up our experiments on Microsoft Azure, it was necessary to purchase a pay-as-you-go subscription for \$29/month. Azure limits a free trial user to 4 cores, while we needed about 150 cores to run the benchmarks. Unfortunately, this took about a month to get funding for from JMU, but it eventually came and we were able to start getting the benchmarks ready. We used [the SLURM cluster template](#) to get our SLURM cluster up and running. If you click on the "Deploy to Azure" link on that page you will be brought to a screen that looks like this:



The screenshot shows a deployment form for a SLURM cluster on Microsoft Azure. It includes fields for Subscription (Pay-As-You-Go), Resource group (Select a resource group), Location (East US), and a SETTINGS section with Admin Username, Admin Password, Vm Size (Standard_D1_v2), Scale Number (2), _artifacts Location (https://raw.githubusercontent.com/Azure/azure-quickstart-templates/master/slurm), _artifacts Location Sas Token, and Location ([resourceGroup().location]).

* Subscription	Pay-As-You-Go
* Resource group	Select a resource group Create new
* Location	East US
SETTINGS	
* Admin Username	
* Admin Password	
Vm Size	Standard_D1_v2
Scale Number	2
_artifacts Location	https://raw.githubusercontent.com/Azure/azure-quickstart-templates/master/slurm
_artifacts Location Sas Token	
Location	[resourceGroup().location]

Figure 1: Microsoft Azure SLURM Template

For the Vm Size input we ended up using Microsoft's Standard_D8s_v3, which is an 8 core machine with hyperthreading available so that we could best represent the local JMU cluster. We would then specify the Scale Number to be 16 in order to launch a login node and 16 compute nodes, each with 8 cores.

A quota increase must be requested to access the appropriate number of cores to launch this cluster. Luckily, the quota increase requests were granted within 4 hours of sending them each time we sent them. We then had to edit the slurm.conf file located at the path /etc/slurm. Permissions had to be changed first with chmod to edit, then we changed the CPUs at the bottom from 1 to 8 to let SLURM know it has access to 8 cpus on each node. In order for this to register it is necessary to restart SLURM, so we needed to add a reboot program by adding the line "RebootProgram=/sbin/reboot" to the slurm.conf file. We next had to run the following commands to restart SLURM and set the node states to idle:

```
sudo scontrol reconfigure
sudo scontrol reboot
scontrol update nodename=master,worker[0-15] state=idle
```

Using the sinfo command SLURM should show a change from some kind of non-idle state to the idle state as the following depicts:

```

tremansibj@master:~/ubuntu-npb/NPB3.3.1/NPB3.3-MPI/bin$ sinfo --long -N
Tue Apr 16 18:59:52 2019
NODELIST      NODES PARTITION      STATE CPUS  S:C:T MEMORY TMP_DISK WEIGHT FEATURES REASON
master        1  debug*      drained  8  8:1:1    1      0      1 (null) Low socket*core*thre
worker0       1  debug*      drained  8  8:1:1    1      0      1 (null) Low socket*core*thre
worker1       1  debug*      drained  8  8:1:1    1      0      1 (null) Low socket*core*thre
worker2       1  debug*      drained  8  8:1:1    1      0      1 (null) Low socket*core*thre
worker3       1  debug*      drained  8  8:1:1    1      0      1 (null) Low socket*core*thre
worker4       1  debug*      drained  8  8:1:1    1      0      1 (null) Low socket*core*thre
worker5       1  debug*      drained  8  8:1:1    1      0      1 (null) Low socket*core*thre
worker6       1  debug*      drained  8  8:1:1    1      0      1 (null) Low socket*core*thre
worker7       1  debug*      drained  8  8:1:1    1      0      1 (null) Low socket*core*thre
worker8       1  debug*      drained  8  8:1:1    1      0      1 (null) Low socket*core*thre
worker9       1  debug*      drained  8  8:1:1    1      0      1 (null) Low socket*core*thre
worker10      1  debug*      drained  8  8:1:1    1      0      1 (null) Low socket*core*thre
worker11      1  debug*      drained  8  8:1:1    1      0      1 (null) Low socket*core*thre
worker12      1  debug*      drained  8  8:1:1    1      0      1 (null) Low socket*core*thre
worker13      1  debug*      drained  8  8:1:1    1      0      1 (null) Low socket*core*thre
worker14      1  debug*      drained  8  8:1:1    1      0      1 (null) Low socket*core*thre
worker15      1  debug*      drained  8  8:1:1    1      0      1 (null) Low socket*core*thre

```

Figure 2: Nodes in drained state

```

tremansibj@master:~/ubuntu-npb/NPB3.3.1/NPB3.3-MPI/bin$ sinfo --long -N
Tue Apr 16 19:00:17 2019
NODELIST      NODES PARTITION      STATE CPUS  S:C:T MEMORY TMP_DISK WEIGHT FEATURES REASON
master        1  debug*      idle    8  8:1:1    1      0      1 (null) none
worker0       1  debug*      idle    8  8:1:1    1      0      1 (null) none
worker1       1  debug*      idle    8  8:1:1    1      0      1 (null) none
worker2       1  debug*      idle    8  8:1:1    1      0      1 (null) none
worker3       1  debug*      idle    8  8:1:1    1      0      1 (null) none
worker4       1  debug*      idle    8  8:1:1    1      0      1 (null) none
worker5       1  debug*      idle    8  8:1:1    1      0      1 (null) none
worker6       1  debug*      idle    8  8:1:1    1      0      1 (null) none
worker7       1  debug*      idle    8  8:1:1    1      0      1 (null) none
worker8       1  debug*      idle    8  8:1:1    1      0      1 (null) none
worker9       1  debug*      idle    8  8:1:1    1      0      1 (null) none
worker10      1  debug*      idle    8  8:1:1    1      0      1 (null) none
worker11      1  debug*      idle    8  8:1:1    1      0      1 (null) none
worker12      1  debug*      idle    8  8:1:1    1      0      1 (null) none
worker13      1  debug*      idle    8  8:1:1    1      0      1 (null) none
worker14      1  debug*      idle    8  8:1:1    1      0      1 (null) none
worker15      1  debug*      idle    8  8:1:1    1      0      1 (null) none

```

Figure 3: Nodes back in idle state

Now with SLURM set up we had to get the benchmarks onto each node and install the necessary libraries to run them, which are gfortran and OpenMPI. To do this we first saved a folder with the benchmarks on our student server. We then used scp to copy them into the login node's home directory. After, we used a script to copy them onto each worker node which can be found with our code. If the code is not compiled yet, it should be compiled before being sent to each worker node. There is a script to do this in our code. We then ran a script to install the necessary libraries on each node which can also be found with our code. At this point we had to verify that SLURM was working and could distribute the tasks to different nodes. In order to do this, we ran commands such as the following and verified that more than a single node is being printed:

```
worker0
worker0
[armanibj@master:~/ubuntu-npb/NPB3.3.1/NPB.3-MPI/bin$ srun -n32 mpirun hostname
master
master
master
```

Figure 4: Checking SLURM scheduling with srun

```

master
master
master
[armanibj@master:~/ubuntu-npb/NPB3.3.1/NPB3.3-MPI/bin$ mpirun -np 64 hostname
master
master
master
master
master

```

Figure 5: Checking mpirun scheduling

At this point we were ready to run the tests. We did so by writing a script that used commands such as the following to print our results:

```

arnan@bj-master: /ubuntu-npb/NPB3.3-1/NPB3.3-MPI/bin$ salloc -n 16 mpirun ./cg.C.16 > ../results/cg/"cg.C.16.$(date +%F_%H).txt"
salloc: Granted job allocation 32
libibverbs: Warning: couldn't open config directory '/etc/libibverbs.d'.
libibverbs: Warning: couldn't open config directory '/etc/libibverbs.d'.
libibverbs: Warning: couldn't open config directory '/etc/libibverbs.d'.
libibverbs: Warning: couldn't open config directory '/etc/libibverbs.d'.
libibverbs: Warning: couldn't open config directory '/etc/libibverbs.d'.
libibverbs: Warning: couldn't open config directory '/etc/libibverbs.d'.
libibverbs: Warning: couldn't open config directory '/etc/libibverbs.d'.
libibverbs: Warning: couldn't open config directory '/etc/libibverbs.d'.
libibverbs: Warning: couldn't open config directory '/etc/libibverbs.d'.
libibverbs: Warning: couldn't open config directory '/etc/libibverbs.d'.
libibverbs: Warning: couldn't open config directory '/etc/libibverbs.d'.
libibverbs: Warning: couldn't open config directory '/etc/libibverbs.d'.
libibverbs: Warning: couldn't open config directory '/etc/libibverbs.d'.
libibverbs: Warning: couldn't open config directory '/etc/libibverbs.d'.
libibverbs: Warning: couldn't open config directory '/etc/libibverbs.d'.
libibverbs: Warning: couldn't open config directory '/etc/libibverbs.d'.
libibverbs: Warning: couldn't open config directory '/etc/libibverbs.d'.
[master:05969] 15 more processes have sent help message help-mpi-btl-base.txt / btl:no-nics
[master:05969] Set MCA parameter "orte_base_help_aggregate" to 0 to see all help / error messages
salloc: Relinquishing job allocation 32
salloc: Job allocation 32 has been revoked.
```

Figure 6: Running a benchmark on Azure

We were unable to fix the warnings depicted in this image before our subscription expired, but they did not seem to have a large affect on the timings of the benchmarks.

2.2 Amazon Web Services

In order to set up our experiments on Amazon Web Service's, we required a limit increase of the particular nodes we used (m2.4xlarge). AWS limits a free user to 2 instances of the m2.4xlarge, which has 8 cores each, when we needed 16 instances totaling to 128 cores to run the benchmarks(including a 4 core login node). Unfortunately, even though Amazon grants you a particular limit for a particular instances, for example we have a 16 instance limit for the m2.4xlarge instances, there is a chance that you may not be allocated the instances if they are in use by someone else. This caused us major confusion as there is no error message and we were unable to get to the bottom of this issue until recently when a Representative from AWS Technical Support emailed us to shed light on the subject. Because of this we were unable to at the time of testing get all 16 instances in use in our cluster which meant we were unable to test the 128 mpi-task benchmarks. To launch our HPC we used [this template](#). From this link you should select "Create the HPC environment"(assuming you have created a key pair already) and then select one of the three regions. This will take you to the "Create Stack" menu in AWS, from here you select next and must give a name and password(this password will not be used to login, it is for NICE engine third party job scheduler, but we will be using SLURM to mimic our cluster environment).

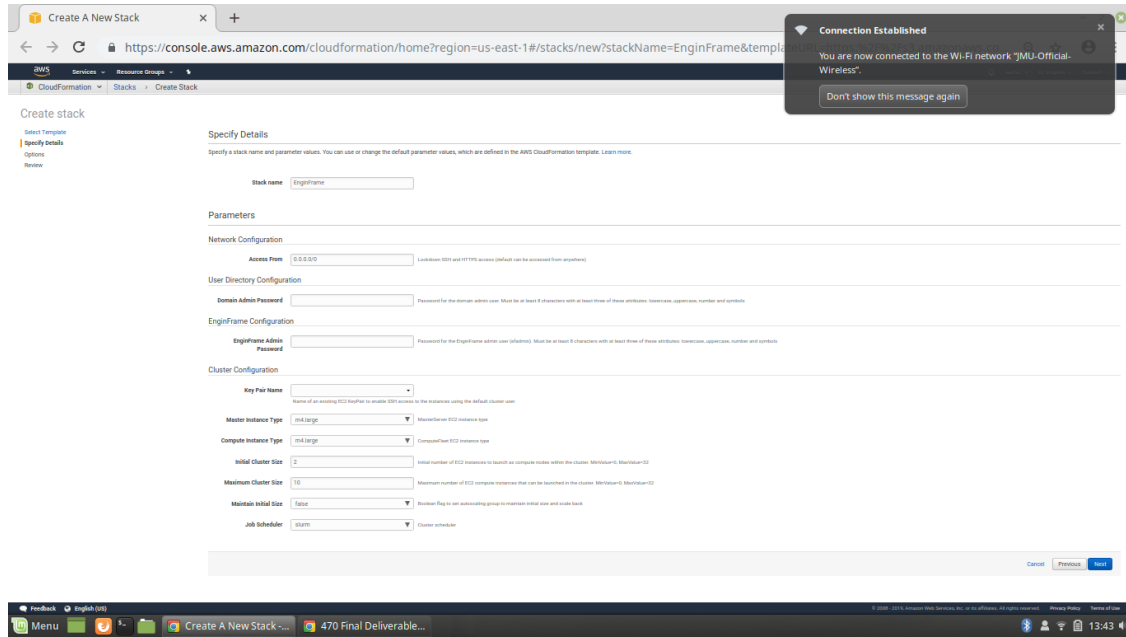


Figure 7: AWS Deployment Template

On this page you can specify the type of instance you want for the login/master node, the compute nodes, and the type of job scheduler you wish to use. Once everything meets your requirements you can select next on the rest of the pages and accept their terms and conditions and create the stack. This will start the creation of your HPC and should take about 45 minutes to an hour for the status to change to "Create Complete". Once you see that status you should be able to ssh to your login node and run the command "module load mpi". After this you should have a HPC cluster up and running for MPI and OpenMP. You can run the "sinfo" command to see what instances are connected, keeping in mind that AWS may allocate nodes for you even if they are not available.

2.3 GCP

In order to run the benchmarks on GCP we required a limit increase on the number of cores (CPU's) from 20 to 132+. We requested a quota increase to 150 for the US-EAST region and it was granted within 3 hours. We used the files from [this GitHub repository](#) with a slightly modified yaml file. Once the yaml has been run, the machines start as pictured and the login node can be connected to by pressing the ssh button and spawning a new window, or providing a valid ssh key via terminal to the provided external IP.

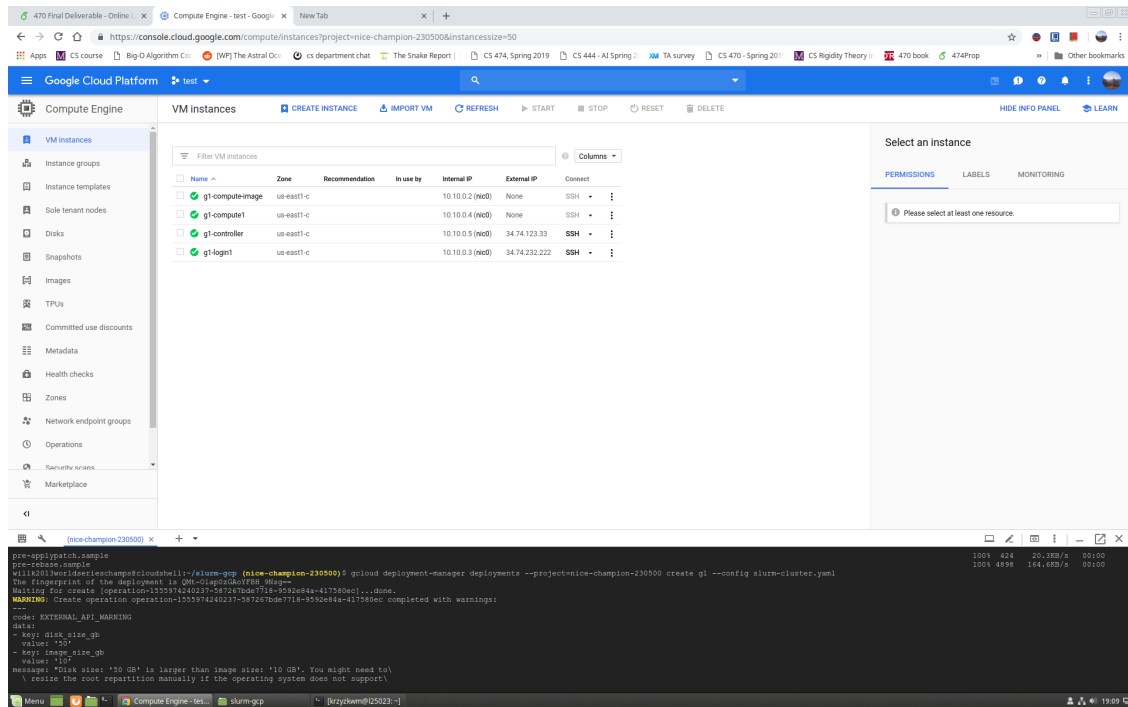


Figure 8: Deployed SLURM Cluster

The only changes to the yaml were providing a username, zone, region, machine types, and count. We then waited for SLURM to install, which took about 20 minutes. Afterwards the login screen looked like this.

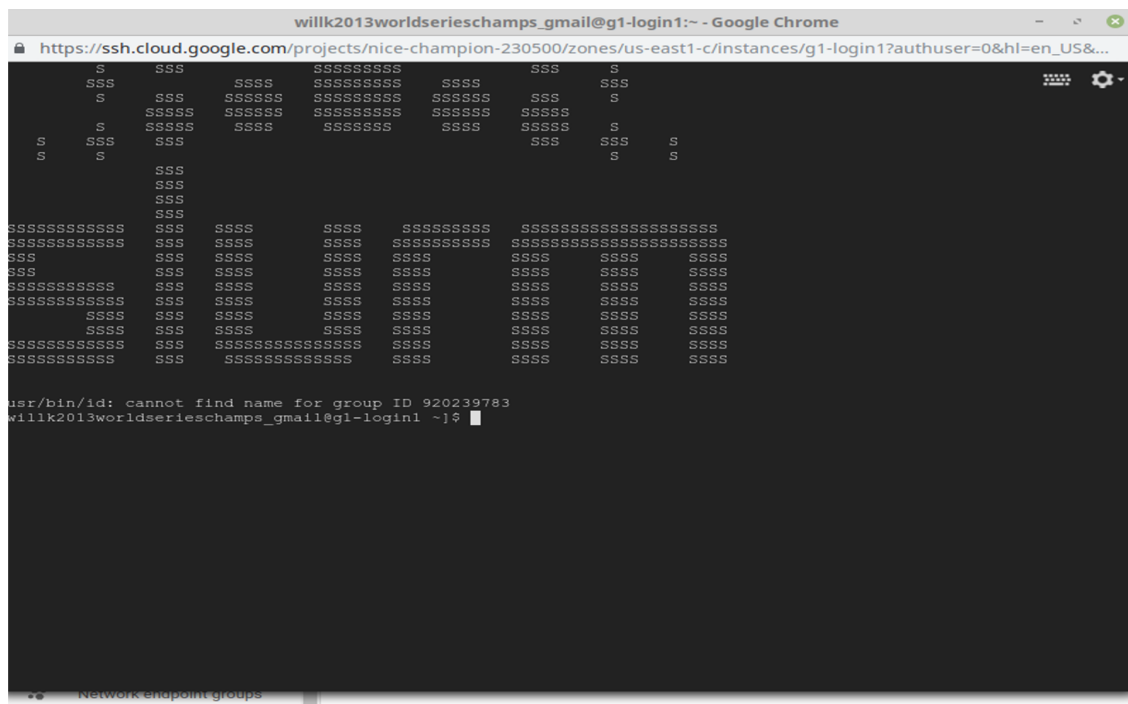


Figure 9: SLURM Login Screen

We then uploaded the test suite and custom shell script files. It appeared that SLURM was not using all 8 cores per machine when running OMP tests, however this was rectified in the shell scripts by using a SLURM command. The GCP machines used [Intel Xeon E5 V3 \(us-east1-c\)](#) cpu’s with hyper threading, which is the same platform [our cluster uses](#).

3 Experiments

We used the NAS benchmarks provided [here](#). We ran the following on all platforms in both MPI and OMP versions.

IS - Integer sorting, random memory access

EP - embarrassingly parallel, baseline

CG - Conjugate Gradient, irregular memory access and communication.

MG - Multi-grid meshes, memory intensive, long and short communication

LU - Lower upper Gaussian solver. Computationally expensive.

For GCP and AWS, we compiled locally on the JMU cluster by removing some unsupported compiler flags and then relying on binary compatibility between RHEL7 and CentOS. For Azure we had to recompile as the OS was Ubuntu 16 and we had cross compatibility issues. When the cluster supported autoscaling we also observed the usage statistics to predict the price of using clusters on each cloud service.

4 Results

4.1 Performance Results

In terms of performance, we mainly evaluated the Lower-Upper Gauss-Seidel solver (LU) C test and the Conjugate Gradient (CG) C tests to evaluate the three cloud services and the cluster. The LU tests use OMP to evaluate computational performance and the CG tests use MPI to evaluate communication performance.

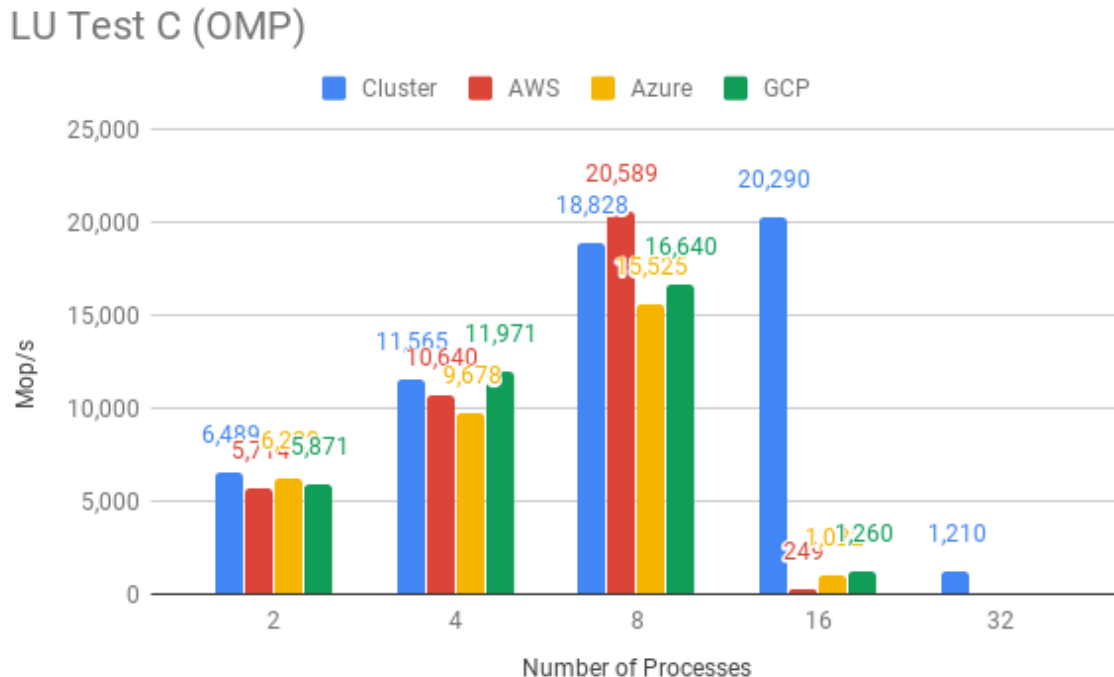


Figure 10: LU Test C on each platform

The LU tests show that the cluster surpasses every cloud cluster in computation power. Although the mop/s is less than AWS for 8 threads, the cluster is far superior with 16 threads, and it was the only distributed system that we were able to get a result for with 32 processes. The cloud services are relatively close in terms of computational ability for the virtual machines that we used, with AWS running more efficiently at 8 processes but far less efficiently at 16 processes.

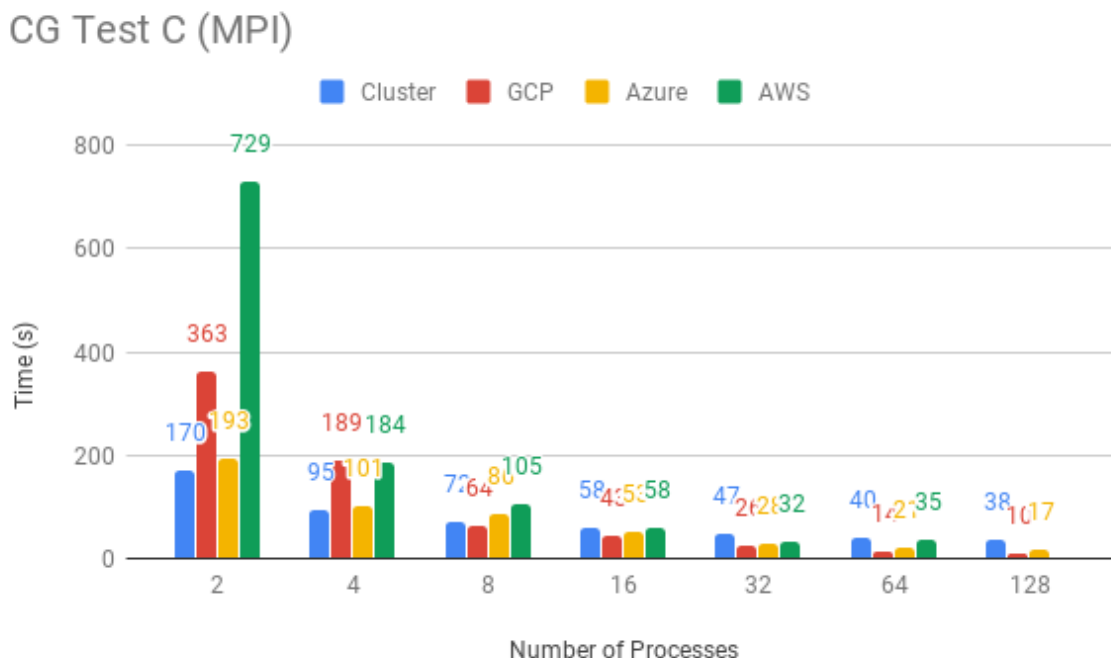


Figure 11: CG Test C on each platform

Surprisingly, the CG tests show that the cloud clusters are superior in communication performance. This holds particularly true for Azure and GCP. The cluster is faster when communicating on the same node, but once inter-node communication is necessary when we reach 16 processes the cloud services are faster. The graph shows that as we add more processes the cloud services beat the cluster by larger and larger amounts. This is an interesting result and is contrary to what we predicted would happen. We thought that the cloud services would be slower in communication since the virtual machines the services gave us theoretically could have been far apart in the data center. On the other hand, we know that the cluster is optimized for communication between the compute nodes with a dedicated bus. However, it appears that the virtual machines from the cloud services were highly efficient in communicating. We still cannot say if this is guaranteed to be the case every time virtual machines are requested, but in our case the machines communicated more effectively than our cluster. Furthermore the poor communication seen on the same node could possibly be mitigated by purchasing "memory optimized" machines.

These were the main two tests that we analyzed. There are other tests included in the NAS Parallel Benchmarks suite, but we felt that these tests were the most meaningful and covered the aspects of performance that we wanted to for this project. We do have the results from the other tests that could be used in the future for further analysis. All data is available on the cluster in our scratch folder

4.2 Cost Results

Our goal for the cost analysis was to find out how much it would cost to run CS470 on the cloud. We acknowledge that the cluster usage is more than just CS470 students as it is used for research and other projects. The cost of the cluster was \$49,452. If the cluster lasts the projected 8 years, then because 470

is taught once per year, one semester costs \$6,181.1. This does not account for any other cluster usage throughout the year. We found that the cluster was used for 9066.38 hours last semester from 1/1/18 - 5/4/18 and it was idle for 367946.68 hours last semester. The cluster uses 16 compute nodes that have 8 cores each, making 128 cores. We can then find how many hours each core would be used:

$$\text{Usage per machine: } 9066.38/16 = 566.65 \text{ hrs/machine}$$

$$\text{Usage per core: } 566.65/8 = 70.831 \text{ hrs/core}$$

We can also find the idle time for each core:

$$\text{Idle Time per core: } 367946.68/16/8 = 2874.58 \text{ hrs/core}$$

Using this information, we were able to calculate how much each cloud service would cost per semester. If possible we utilized usage data from each cloud service while running our tests to estimate the cost of every core of every machine for the full 70.831 hours of its usage. This would be the worst case for how much the cluster on the cloud would cost. For GCP, we found that the 16 machines would cost \$10 per hour with the 2019 rates. In GCP we were able to utilize autoscaling which removes the nodes we are not using so that we do not have to pay for them and adds them back in when needed. We found the cost of idle time with this feature to be about \$0.61 per hour. This means the total cost for GCP per semester would be:

$$\text{Total: } \$10 * 70.831 \text{ hours} + \$0.61 * 2874.58 \text{ hours} = \$2461.80 \text{ per semester}$$

For Microsoft Azure we were unable to use autoscaling with our SLURM template. For the cost of idle time we instead used their pricing calculator. We assume there would 3 nodes reserved when the cluster is idle. This yielded the following result assuming 718 hours of idle time per month:

The screenshot shows the Microsoft Azure Pricing Calculator interface. The configuration is as follows:

- REGION:** East US
- OPERATING SYSTEM:** Linux
- TYPE:** Ubuntu
- TIER:** Standard
- INSTANCE:** D8s v3: 8 vCPU(s), 32 GB RAM, 64 GB Temporary storage, \$0.384/hour
- Billing Option:** Pay as you go (selected), 1 year reserved (~40% discount), 3 year reserved (~62% discount)
- Quantity:** 3 Virtual machines
- Hours:** 718
- Total:** \$827.14 Per month
- Managed OS Disks:** Tier: Standard HDD

Figure 12: Microsoft Azure Cost Calculator

We estimated the cost of using all cores of the cluster to be \$9.25 per hour. Using these numbers, we found estimated Microsoft Azure's cost per semester:

$$\text{Total: } \$9.25 * 70.831 \text{ hours} + \$827.14 * 4 \text{ months} = \$3963.75 \text{ per semester}$$

For AWS we were able to take advantage autoscaling with the given template. Using the AWS cost calculator we estimate the cost of 3 idle instances of the m2.4xlarge would cost about \$2.94 an hour. For amazon using 16 instances would cost \$15.68 per hour at the current rates. Since we were able to auto scale as well our total cost per semester would be as follows:

$$\text{Total: } \$15.68 * 70.831 \text{ hours} + \$2.94 * 2874.58 \text{ hours} = \$9561.89 \text{ per semester}$$

5 Discussion

5.1 Ease of use

Part of this project was to determine whether it would be beneficial to run CS 470 in the cloud instead of the cluster. One suggested avenue was Kubernetes, which is supported by all the cloud platforms. However, this was quickly discarded as there is no easy way to run a "job" equivalent; this requires the use of images and containers and was deemed too much of a learning curve. Instead we used various Git repositories endorsed by the platforms. Some worked, some were unreadable by the current platform, and some required additional installs such as FORTRAN. The main result from all of this however, has been that any class hosted in the cloud would be very susceptible to platform changes. If GCP were to modify how the deployment manager works for example, the entire yaml file would be worthless. In fact this seems to have happened to several AWS templates we attempted to use. Currently the best option is GCP, simply due to ease of use and the exceptional yaml that has been commissioned by Google from the makers of SLURM. However GCP does not have the best performance, failing to match the cluster on high thread counts and low process counts. It is possible that the performance could be improved by purchasing better machines, especially if there is more room in the budget from moving to a cloud service.

5.2 Cost

GCP is far cheaper than the cluster, not even accounting for power consumption or maintenance. We could use some of the savings to pay for more expensive hardware, such as a different cpu platform or more memory. However, the estimate for the cluster is old, and new hardware may be cheaper. Microsoft Azure was more expensive than GCP, but still significantly cheaper than the cluster is per semester. Unfortunately the Azure template did not allow for auto scaling, and so the cost is more expensive if that is not implemented somehow. Azure also had slightly better performance for hyper threading OMP tests than AWS. Furthermore Azure and GCP had no issues satisfying our computer requests. The fact that AWS was so much more expensive suggests that Azure and GCP may be operating at a loss currently to gain market share. In our opinion these prices are highly subject to change in the future.

5.3 Performance

We found that the cloud performed better for communication bound programs and the cluster performed better for computation bound programs. This result was contradictory to what we originally thought and gives a new perspective on comparing the cloud systems and the cluster. It should be noted that the cloud systems have more room to improve computational performance because one can easily purchase better hardware. GCP performs better on average than the other cloud providers for threading tests, although it still fails miserably at hyper-threading compared to our cluster. For MPI processes, GCP performs poorly on same node communication, however this may be mitigated with better memory. However GCP rapidly overtakes all competition for inter-node communication. Azure performs very similarly to the cluster for same node MPI communication, and often halves inter node communication. Azure is however middle of the pack for OMP threading tests.

6 Conclusion

It would indeed be cheaper to run 470 on the cloud than on our own dedicated cluster. However, this requires giving up a lot of flexibility and control that does not have monetary value. We have seen in recent years that AWS has had outages and other providers may as well. Furthermore, there is no guarantee that starting the cluster would work every semester, as platform updates may invalidate options without warning. Each semester may also see different performances as you cannot request the same machines each time. If we do use auto-scaling, there is no guarantee that the nodes could be requested, and even if we do get them they require set up time. Moving to the cloud requires many trade offs and does not have a clear cut answer. Given our findings, we do not think that it is necessary to move CS 470 to the cloud. Although the cloud has its advantages over the cluster, the differences are not significant enough for the move to be worth it at this time. When the cluster's life comes to an end this question should be revisited under those circumstances, but we do not believe it is worth replacing outright. The cloud is difficult to set a SLURM cluster up on and would be hard to manage with many students using it simultaneously. The control of having our own cluster is also preferable to being at the mercy of the cloud service provider we would be using.

7 Future Work

[This repository](#) was suggested by Dr. Lam. Unfortunately, by the time it was brought to our attention there was little time to explore it. Furthermore, we have collected a wealth of data that the three of us are unable to adequately analyze in the time left. Finally, a more adequate cost comparison can be made if the cost for a new JMU HPC was explored. In the future the cost for a new cluster may be less than our estimates as semi-conductor prices decrease, so a new cost estimate for the cluster will need to be evaluated when our cluster becomes obsolete/outdated.

8 References

SLURM cluster. (n.d.). Retrieved April 25, 2019, from <https://azure.microsoft.com/en-us/resources/templates/slurm/>
NAS Parallel Benchmarks. (n.d.). Retrieved from <https://www.nas.nasa.gov/publications/npb.html>
Regions and Zones — Compute Engine Documentation — Google Cloud. (n.d.). Retrieved from <https://cloud.google.com/compute/zones>
How to launch an HPC environment on Amazon Web Services - AWS. (n.d.). Retrieved from <https://aws.amazon.com/getting-started/use-cases/hpc/>
CS 470. (n.d.). Retrieved from https://w3.cs.jmu.edu/lam2mo/cs470_2019_01/cluster.html